《软件系统设计报告》

项目名称	无人机-无人车协同柑橘采摘系统
系统名称	控制与决策子系统
专　　业	计算机科学与技术学院 / 软件工程
学　　号	2327406014
姓　　名	朱金涛
日　　期	2025 年 12 月 19 日
版　　本	V1.0

目录

1. 引言

1.1. 项目背景与意义

随着现代农业向智慧化、无人化方向迈进，果园采摘作为农业生产中**耗时最长、用工最多**的环节，正面临着严峻的“用工荒”与成本上升挑战。传统的单一农业机器人受限于感知范围与作业效率，难以在复杂的非结构化果园环境中实现高效作业。

在此背景下，“空地协同”技术应运而生。利用无人机（UAV）的高空广域感知优势与无人车（UGV）的地面负载作业优势，构建**异构多机器人协同系统**，成为提升采摘效率、降低运营成本的关键路径。本项目的研究意义在于探索并实现一套高效、稳定的“控制与决策子系统”，作为协同采摘系统的“中枢大脑”，解决多智能体协作中的任务调度、路径规划及数据融合难题，为**智慧农业**的落地提供工程实践参考。

1.2. 国内外研究现状

目前，农业机器人的研究主要集中在单体智能方面，如单一的采摘机械臂或巡检车。虽然在目标识别与抓取技术上取得了一定突破，但在多机协同调度领域仍处于起步阶段。

- **国外现状**：欧美等国在精准农业领域较早引入了多机器人系统（Multi-Robot Systems），利用 ROS（机器人操作系统）实现了初步的编队控制，但在应对复杂果园地形时的动态任务分配策略上仍有优化空间。
- **国内现状**：我国在农业无人机应用方面处于领先地位，但空地协同技术多用于安防或测绘，在农业采摘场景下的应用尚少。现有的调度系统往往**缺乏对异构数据**（如高频遥测与大容量图像）的高效融合处理能力。

1.3. 本文工作内容

本文以“无人机-无人车协同柑橘采摘系统”为背景，重点设计并实现了其中的核心模块——**控制与决策子系统**。整个开发过程遵循软件工程全生命周期规范，主要工作内容如下：

1. **需求分析与建模**：基于已完成的《软件需求规格说明书》（SRS）的编写，利用 UML 对核心业务进行了用例建模与动态交互分析。
2. **系统架构设计**：针对系统的实时性与扩展性需求，设计了基于“分层架构 + 发布/订阅模式”的混合软件架构，并应用了策略模式与单例模式优化内部逻辑。
3. **全栈开发与实现**：基于 Python/FastAPI 构建了后端控制服务，设计了 PostgreSQL + PostGIS 的空间数据库方案，实现了三维果园地图的数据持久化与数字孪生前端展示。
4. **工程过程管理**：引入华为云 CodeArts 平台，建立了“需求规划-代码托管-在线建模”的 DevOps 闭环，保证了开发过程的规范性与可追溯性。

2. 概要设计

2.1. 架构风格选择与说明

针对“无人机-无人车协同柑橘采摘系统”的业务特性与非功能需求，本系统选用了 **混合架构风格**，具体由以下三种风格组合而成：

2.1.1. 分层架构：

说明：将系统纵向划分为决策层、任务层、世界模型层和基础设施层。

理由：本系统涉及复杂的硬件控制与高层智能决策。分层架构能够有效地实现“关注点分离”，将底层的传感器通信细节（基础设施层）与上层的任务调度逻辑（任务层）解耦。这符合 SRS 中关于“模块化设计” 和“层次化决策结构” 的要求。

2.1.2. 发布-订阅模式：

说明：系统各构件之间，以及系统与外部 UAV/UGV 之间，通过消息总线进行异步通信。

理由：系统必须遵循 ROS 2 标准 ，且需处理多源异构数据（如 10Hz 的图像数据和 50Hz 的车辆状态 ）。发布-订阅模式支持松耦合的异步通信，能够满足高并发数据处理和分布式控制的需求。

2.1.3. 微内核/插件模式：

说明：核心的任务分配引擎仅定义接口，具体的分配算法作为可插拔策略实现。

理由：SRS 明确要求系统具备“可扩展性设计”，且任务分配需支持多目标优化（如能耗优先、时间优先）。插件模式允许在不修改核心代码的情况下，动态替换或新增分配算法。

2.2. 系统架构图

下图展示了“控制与决策子系统”的逻辑包图，描绘了系统的静态结构及包之间的依赖关系。

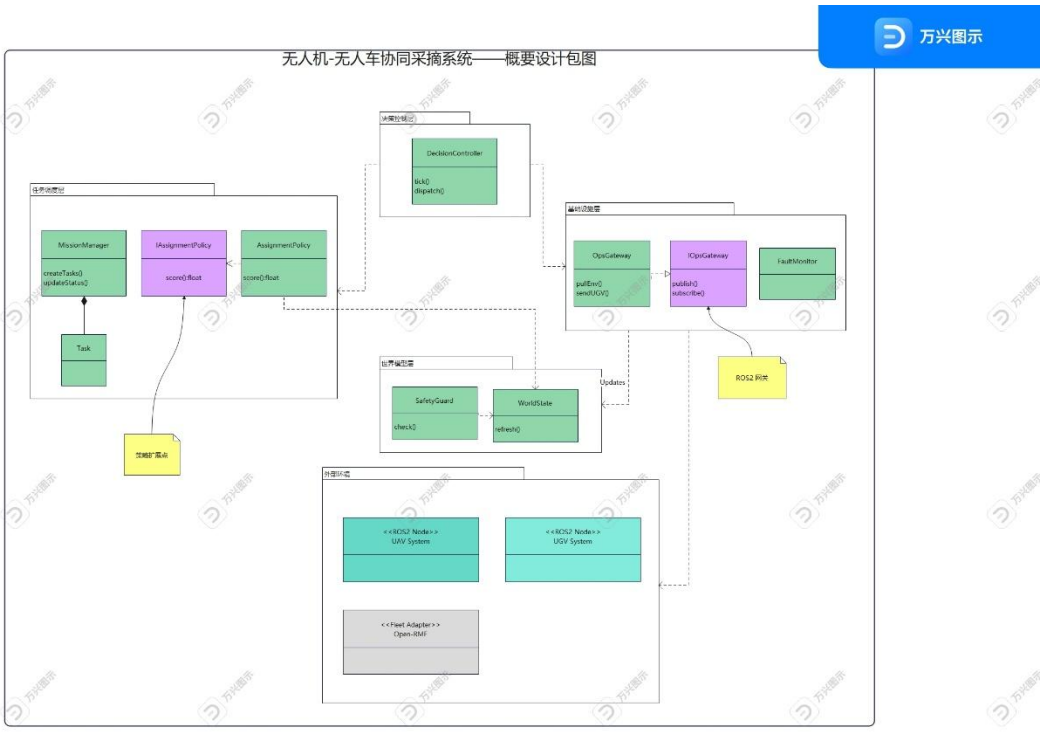

图 1 包图

2.2.1. 架构图解释说明

系统自上而下划分为四个核心逻辑包，依赖关系严格遵循单向依赖原则（上层依赖下层），避免循环依赖：

2.2.1.1. 决策控制层

职责：作为系统的顶层入口，负责主控循环 (DecisionController)。它协调各子模块的运行节奏，不包含具体的业务算法，仅负责调度。

关系：依赖任务层进行任务生成，依赖世界模型层获取状态，依赖基础设施层下发指令。

2.2.1.2. 任务调度层

职责：负责全生命周期的任务管理。包含任务管理器 (MissionManager) 和分配引擎 (AssignmentEngine)。该层实现了 IAssignmentPolicy 接口，支持根据 SRS 要求的多种优先级评估模型 进行策略扩展。

关系：依赖世界模型层的数据来评估任务的可行性。

2.2.1.3. 世界模型层

职责：作为数据中心，维护全局的“上帝视角”。它融合了来自 UAV 和 UGV 的多源数据 ，并通过 SafetyGuard 执行严格的安全约束检查（如电量、碰撞检测）。

关系：被上层模块依赖，自身不依赖上层业务逻辑，确保数据的纯净性。

2.2.1.4. 基础设施层

职责：充当防腐层（Anti-Corruption Layer）。通过 OpsGateway 封装底层的 ROS 2 通信细节 ，将外部的硬件信号转换为系统内部的标准对象。

关系：直接与外部环境（ROS 2 节点）交互，并负责更新世界模型层的数据。

2.3. 关键接口定义

基于架构图中的设计，定义以下三个核心接口，以确保系统的模块化协作与可扩展性。

2.3.1. 任务分配策略接口 (IAssignmentPolicy)

所属包：任务调度层 (com.citrus.mission)

功能描述：定义任务分配算法的标准化规范。该接口允许系统根据不同的作业场景（如抢收

模式、节能模式）动态切换底层的优化算法（遗传算法或粒子群算法 ）。

主要方法:

float score(Task, UGVState ugv): 计算特定车辆执行特定任务的匹配度得分（基于距离、电量等因子）。

List<Assignment> compute(List<Task> tasks, WorldState world): 输入待分配任务和当前环境状态，输出最优的分配方案。

2.3.2. 硬件网关接口 (IOpsGateway)

所属包：基础设施层 (com.citrus.infrastructure)

功能描述:定义系统与外部硬件交互的标准契约。该接口屏蔽了具体的通信协议(DDS/TCP)差异，确保系统符合 ISO 11783 标准兼容性要求。

主要方法:

- void publishCommand(UGVCommand cmd): 向指定无人车下发导航或采摘指令。
- void subscribeTelemetry(): 启动对无人机和无人车状态数据的监听。
- EnvSnapshot pullEnv(): 拉取最新的环境感知快照（包含果实位置、障碍物信息）。

2.3.3. 安全校验接口 (ISafetyGuard)

所属包：世界模型层 (com.citrus.world)

功能描述：提供统一的安全检查服务。在任务分配或指令下发前，校验操作是否违反物理约束或安全规则，确保满足“安全与容错功能需求”。

主要方法:

- boolean checkConstraints(AssignmentPlan plan): 校验分配方案是否满足电量阈值、载重限制等硬性约束。
- RecoveryAction recover(AlarmType type): 根据告警类型（如低电量、通信中断），生成标准的应急恢复预案（如返航、原地待命）。

2.4. 系统技术栈选择

为满足 SRS 中对实时性（<100ms 响应 ）、分布式协作及 AI 识别精度的要求，本系统计划采用以下技术栈进行实现：

2.4.1. 操作系统与中间件：

Ubuntu 22.04 LTS：作为基础运行环境。

ROS 2 Humble：利用其 DDS 通信机制实现多机协同的高频数据交换。

2.4.2. 核心算法与调度：

Open-RMF：用于多机器人车队的交通管制与任务调度，解决多车路径冲突问题。

YOLOv11 + PyTorch：实现对柑橘成熟度与位置的高精度实时识别。

Nav2 (Navigation 2)：基于 A* 和 DWA 算法实现无人车的全局路径规划与局部动态避障。

2.4.3. 仿真与开发工具：

Gazebo：构建三维果园仿真环境，进行算法验证。

C++ 17：用于开发核心决策逻辑与底层驱动，确保系统的高性能。

Python 3.10：用于开发上层业务逻辑及 AI 模型接口。

3. 功能迭代设计

基于需求规格说明书中的初步分析模型，本阶段对系统的核心类进行了详细设计与优化迭代。主要工作包括引入设计模式以增强系统的可维护性，以及细化类属性与接口定义以指导具体的代码实现。

3.1. 迭代分析说明

3.1.1. 策略模式的应用（针对任务分配功能）

分析：原设计中任务分配逻辑硬编码在 MissionManager 中，难以适应未来新增的“抢收模

式”或“夜间模式”需求。

迭代：将分配算法抽离为 IAssignmentPolicy 接口（详见类图）。AssignmentEngine 类不再依赖具体的算法，而是持有该接口的引用。

效果：符合开闭原则（OCP），新增分配策略时无需修改现有代码，仅需新增一个实现类即可。

3.1.2. 单例模式的应用（针对环境状态管理）

分析：WorldState 存储了所有 UAV 和 UGV 的实时状态，如果系统中存在多个实例，可能导致数据不同步，引发严重的决策冲突（如两车抢占同一位置）。

迭代：将 WorldState 设计为线程安全的单例（Singleton）。

效果：确保系统内任何模块（如任务层、基础设施层）访问的都是同一份最新的环境快照。

3.1.3. 对象属性的精确化

分析：早期模型仅定义了业务名词。

迭代：为属性增加了具体的 C++/Java 数据类型定义。例如，将任务列表定义为 List<Task>，将车辆映射定义为 Map<String, UGVState>，并为关键状态（如 TaskStatus）引入了枚举类型，提高了类型安全性。

3.2. 详细设计类图

下图展示了迭代后的核心类设计，重点体现了任务调度模块的策略模式结构及单例模式结构。

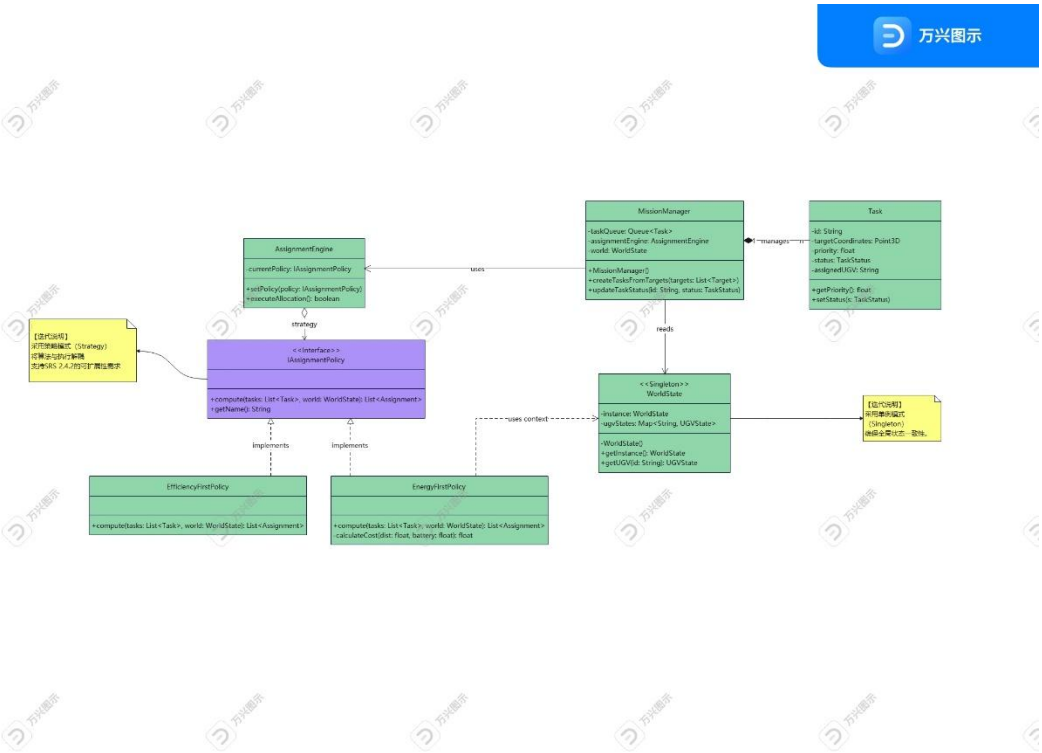

图 2 详细设计类图

4. 界面设计

本系统采用了扁平化的层级导航结构，以降低操作复杂度并提高现场作业效率。界面设计严格遵循“模型-视图-分离”原则，通过可视化组件直观呈现后台 WorldState 中的多源异构数据，并为管理员提供对 MissionManager 调度逻辑的人工干预接口。

4.1. 界面导航设计

系统的交互流程设计以“综合监控大屏”为核心枢纽，向下发散至各功能子模块。用户登录后，默认进入全局监控视角，并可通过侧边导航栏快速切换至任务管理、感知视图及系统设置模块。这种设计确保了关键信息（如车辆位置、告警信息）始终处于最高优先级展示。

具体导航层级逻辑如下：

- **一级层级**：系统登录页 -> 综合监控大屏（首页）。
- **二级层级**：从首页可跳转至“实时感知视图”（查看摄像头数据）、“任务管理中心”（查看队列与策略配置）、“系统设置与日志”（参数调整）。

- **三级层级**：各子模块内部的详细参数配置与历史记录查询。

界面导航结构树状图：

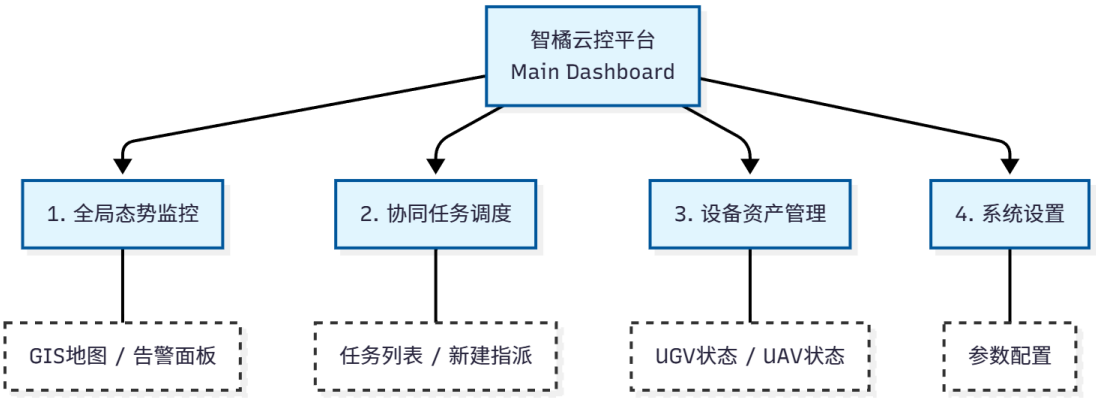

图 3 界面导航树状图

4.2. 关键界面原型设计

根据系统架构设计，本系统界面采用了现代化的 **macOS 玻璃拟态** 设计语言。通过深色背景、半透明磨砂质感与高亮微交互元素，构建了沉浸式的作业环境。本节重点描述“全局态势感知”与“协同任务调度”两个核心交互界面。

4.2.1. 综合监控大屏

对应图片：

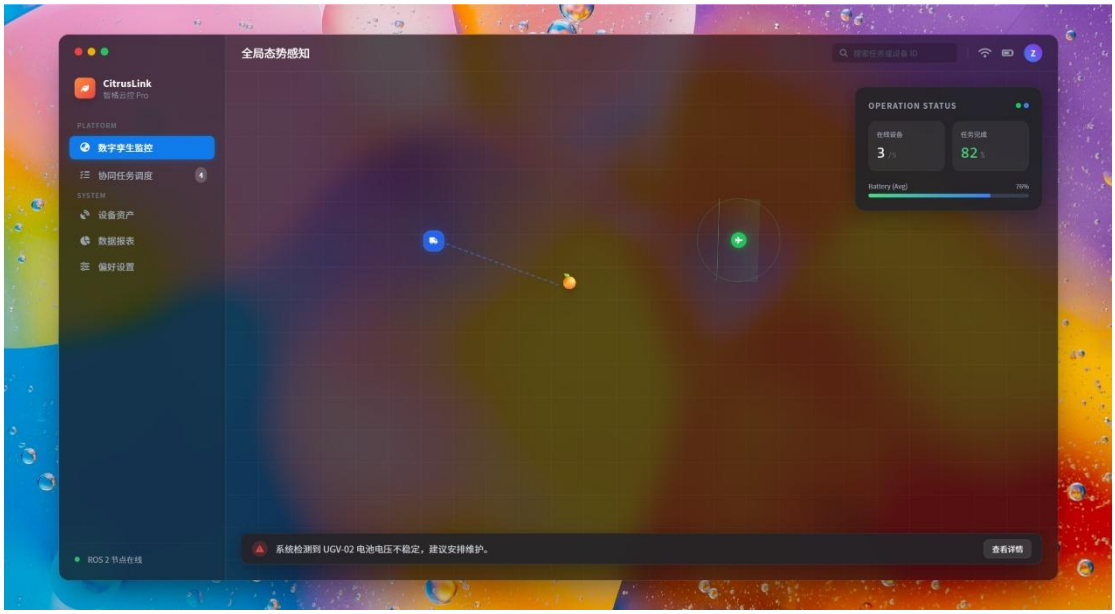

图 4 界面设计效果图-1

设计目的： 该界面作为系统的默认主视图，旨在通过数字孪生技术，将 WorldState 中的抽象数据映射为可视化的果园作业场景。

界面布局与功能细节：

- **侧边导航栏：**
  - 采用高透磨砂玻璃材质，左下角集成 ROS 2 节点在线 状态指示灯（绿色呼吸点），实时反馈底层通信链路的健康状况。
- **中央数字孪生区：**
  - 背景采用深色网格化地图，模拟 SLAM 建图后的空间坐标系。
  - 实体映射：界面实时渲染了协同作业的实体——蓝色图标代表 UGV（无人车），绿色雷达圈图标代表 UAV（无人机）。
  - 动态交互：图标间通过虚线连接，直观展示了当前车辆正在追踪的 Target（目标果实），实现了感知数据的可视化闭环。
- **悬浮状态看板：**
  - 位于界面右上角的悬浮卡片，实时统计关键运维指标。

- 数据可视化：显示“在线设备 (3/5)”与“任务完成率 (82%)”，底部通过蓝绿渐变进度条展示车队平均电量 (76%)，便于管理员一瞥掌握系统健康度。

➢ **智能告警横幅：**

- 位于屏幕底部的深色半透明横幅。
- 异常响应：当 ISafetyGuard 接口监测到异常时（图中显示“系统检测到 UGV-02 电池电压不稳定”），会自动弹出红色警示图标及文字，并提供“查看详情”入口以支持快速排障。

4.2.2. 协同任务调度

对应图片：

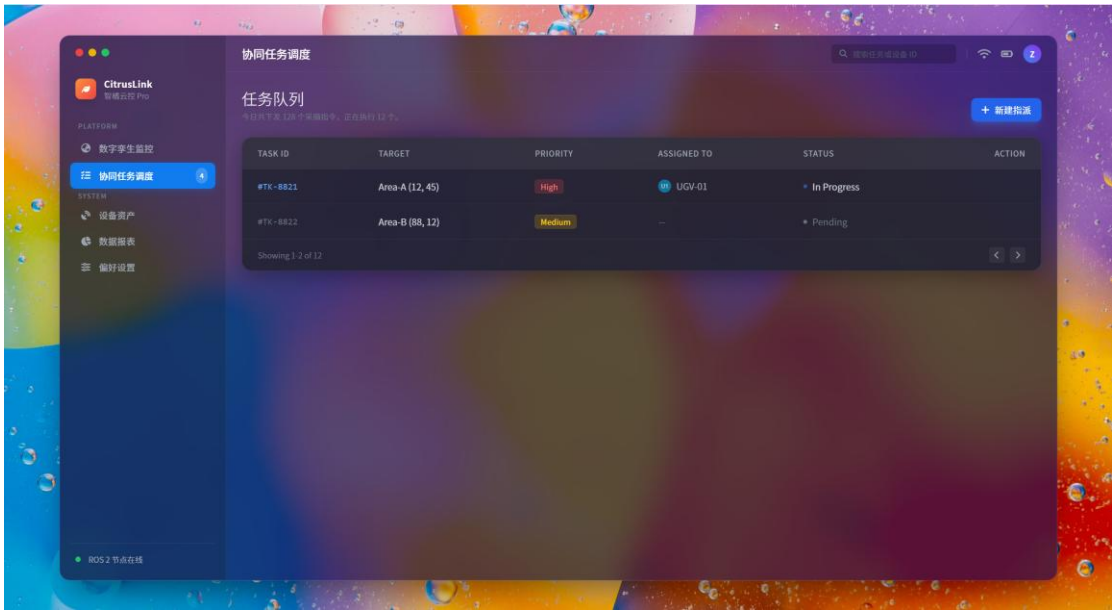

图 5 界面设计效果图-2

设计目的： 该界面用于可视化 **任务调度层** 的运行逻辑 。管理员可在此查看自动化分配结果，并根据实际作业需求，动态干预 **分配引擎** 的决策策略。

界面布局与功能说明：

➢ **任务队列看板**:

- 以看形式展示 TaskQueue 中的任务流转状态，分为“待分配”、“执行中”、“已

完成”三列。

- 每个任务卡片显示任务 ID、目标坐标及优先级评分。

- **策略配置面板**：
  - 提供可视化的策略切换开关，直接映射后台的 **策略模式接口(IAssignmentPolicy)**。
  - **“抢收模式”按钮**：点击后，系统后台自动切换至 EfficiencyFirstPolicy 实现类，优先考虑时间最短。
  - **“节能模式”按钮**：点击后，切换至 EnergyFirstPolicy 实现类，优先考虑能耗最低。
- **人工干预区**：
  - 允许管理员手动暂停特定任务或强制重置某台 UGV 的任务状态，以应对突发情况。

5. 数据库设计

数据库设计是“智橘云控平台”数据持久化的基石。本系统不仅需要存储常规的任务与用户权限数据，还需处理高频的机器人状态遥测数据和复杂的空间地理信息。本章节将从选型、逻辑建模、物理建模及 ORM 映射四个维度展开阐述。

5.1. 数据库选型与说明

经过对系统业务需求（高并发写入、空间计算）的分析，本系统选用 **PostgreSQL 14** 搭配 **PostGIS 插件** 作为核心关系型数据库。

- **原生空间数据支持**：系统核心业务涉及大量三维坐标 (x, y, z) 的处理。PostGIS 提供了高效的 GEOMETRY 类型和 R-Tree 索引，能够直接在数据库层面进行“查询最近车辆”或“判断果实是否在采摘区”的计算，避免了应用层的繁琐运算。
- **高并发遥测处理**：ROS 2 节点会以高频率上报状态数据，PostgreSQL 的 MVCC（多版本并发控制）机制能有效支撑高频写入，防止读写锁造成的阻塞。

5.2. 逻辑模型设计

逻辑设计阶段主要梳理实体及其相互关系。如图所示，本系统包含五个核心实体。

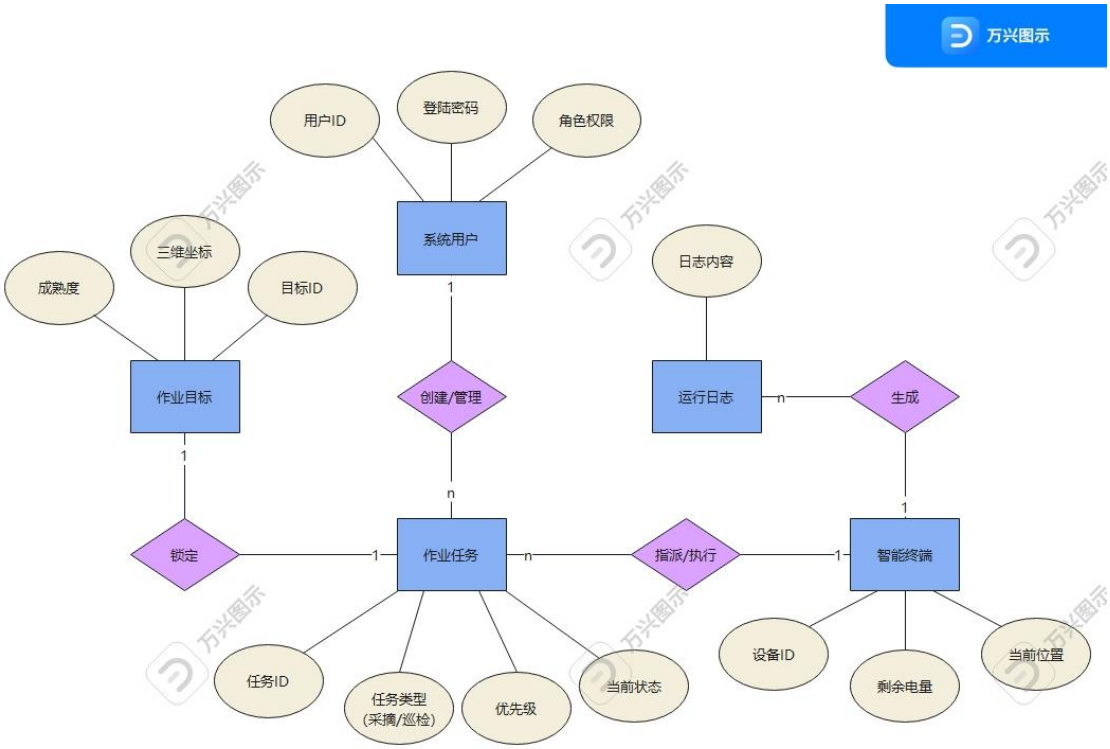

图 6 ER 图

实体与关系说明：

- 管理关系（1:n）：系统用户作为权限主体，可以创建或干预多个作业任务。该关系支撑了系统的人工接管与审计功能。
- 指派关系（n:1）：多个作业任务可以被分派给不同的车辆，但在单一时间点，一个任务只能由一台智能终端（Robot）执行。
- 锁定关系（1:1）：每一个作业任务都精确对应一个且唯一的作业目标，实现了调度逻辑与物理实体的解耦。
- 记录关系（1:n）：为了满足可追溯性，每台智能终端在运行过程中会产生多条运行日，记录报错信息与状态变更。

5.3. 物理模型设计

根据逻辑模型，设计了以下核心数据表。表名遵循 t_模块_名称 规范，主键统一使用 UUID

或自增 ID。

5.3.1. 作业任务表 (t_biz_task)

持久化 MissionManager 中的调度数据，对应 E-R 图中的“作业任务”。

字段名	类型	约束	说明
id	VARCHAR(36)	PK	任务唯一标识 (如 "TK-8821")
priority	INT	NOT NULL	优先级 (0:Low, 1:Medium, 2:High)
status	VARCHAR(20)	NOT NULL	当前状态 (PENDING, IN_PROGRESS, COMPLETED)
type	VARCHAR(20)	-	任务类型 (PICKING/INSPECTION)
created_by	VARCHAR(36)	FK	创建该任务的用户 ID
assigned_robot_id	VARCHAR(36)	FK	执行车辆 ID (关联 t_sys_robot)
target_id	BIGINT	FK, UNIQUE	对应的果实目标 ID (1:1 约束)

字段名	类型	约束	说明
created_at	TIMESTAMP	DEFAULT NOW	创建时间

5.3.2. 智能终端表 (t_sys_robot)

存储所有注册设备的状态信息，对应 E-R 图中的“智能终端”。

字段名	类型	约束	说明
id	VARCHAR(36)	PK	设备 ID (如 "UGV-01")
ip_address	INET	UNIQUE	ROS 2 节点通信 IP
battery_level	FLOAT	CHECK(0-100)	剩余电量百分比
current_load	FLOAT	-	当前载重 (kg)
status	VARCHAR(10)	-	在线状态 (ONLINE/OFFLINE)
last_heartbeat	TIMESTAMP	-	最后一次心跳时间

5.3.3. 作业目标表 (t_biz_target)

存储环境感知快照数据，对应 E-R 图中的“作业目标”。

字段名	类型	约束	说明
id	BIGINT	PK	自增 ID
coordinate	GEOMETRY(POINTZ)	NOT NULL	**三维空间坐标** (PostGIS 类型)
ripeness	FLOAT	CHECK(0-1)	成熟度评分 (YOLO 置信度)
image_url	TEXT	-	视觉识别时的截图路径
area_code	VARCHAR(10)	-	所属区域编号 (如 "Area-A")

5.3.4. 系统用户表 (t_sys_user)

对应 E-R 图中的“系统用户”。

字段名	类型	约束	说明
id	VARCHAR(36)	PK	用户 ID
username	VARCHAR(50)	UNIQUE	登录用户名
password_hash	VARCHAR(255)	NOT NULL	加密后的密码

字段名	类型	约束	说明
role	VARCHAR(20)	-	权限角色（ADMIN/OPERATOR）

5.3.5. 运行日志表（t_sys_log）

对应 E-R 图中的“运行日志”。

字段名	类型	约束	说明
id	BIGINT	PK	自增 ID
robot_id	VARCHAR(36)	FK	产生日志的设备 ID
content	TEXT	-	日志详细内容
level	VARCHAR(10)	-	级别（INFO/WARN/ERROR）
created_at	TIMESTAMP	DEFAULT NOW	发生时间

5.4. ORM 映射策略

为了实现面向对象代码与关系型数据库的解耦，本系统采用了 **Data Mapper** 模式进行 ORM（对象关系映射）设计。

- **映射工具**：后端服务使用 **SQLAlchemy** (Python) 框架。
- **枚举与状态映射**：
  - 代码中的 TaskStatus.IN_PROGRESS 枚举值直接映射为数据库中的字符串

"IN_PROGRESS"，增强了数据库数据的可读性，便于运维排查。

- **空间数据映射**：
  - 利用 **GeoAlchemy2** 库，将数据库中的 GEOMETRY(POINTZ) 二进制流直接映射为代码中的 Point3D 对象。
  - **优势**：开发者可以直接调用 db.session.query(Target).filter(Target.coordinate.ST_DWithin(robot_loc, 5)) 来查询 5 米内的果实，而无需手动编写复杂的 SQL 空间函数。

5.5. 数据库的具体实施

为验证上述逻辑模型与物理模型的正确性，以及 ORM 映射策略在实际工程中的可行性，本阶段编写了后端测试脚本 view_data.py。该脚本基于 SQLAlchemy 框架，对系统核心数据表进行了模拟数据的持久化存储与读取测试，重点验证了 PostGIS 空间扩展对三维坐标的支持能力。

5.5.1. 验证环境与方法

测试环境：WSL2 (Ubuntu 24.04) + PostgreSQL 14 (PostGIS 3.3)。

验证方法：

- 执行 database_setup.py 初始化数据库表结构。
- 执行 populate_data.py 注入包含三维坐标 (x, y, z) 的模拟业务数据（如 admin 用户、UGV 机器人及果实目标）。
- 通过控制台输出日志，检查数据读取的完整性及字段映射的准确性。

5.5.2. 结果展示

下图展示了后端脚本运行后的终端输出日志，验证了系统数据层已具备完整的业务支撑能力。

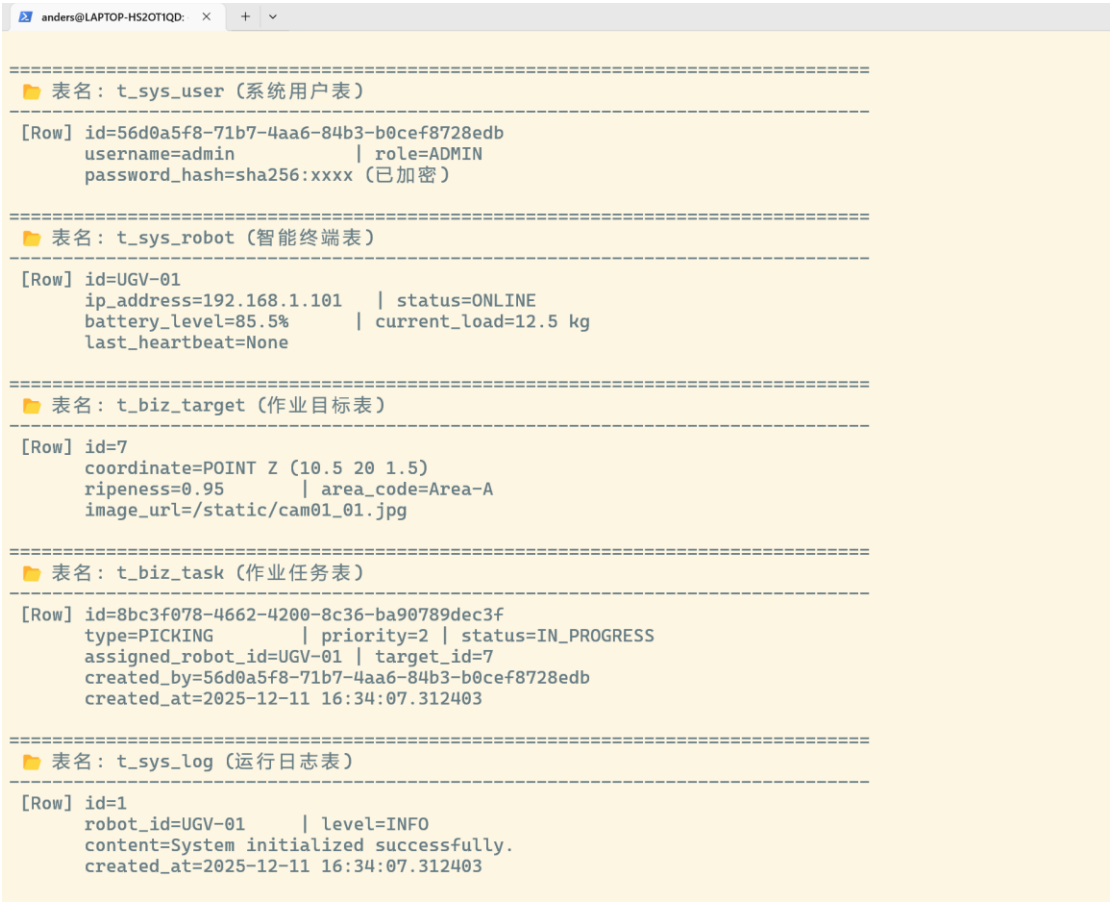

图 7 数据库实现

结果说明：

- **物理约束有效**：t_sys_robot 表正确存储了 INET 类型 IP 地址，且电量与成熟度数值均符合 CHECK 约束范围，验证了数据库级校验机制的有效性。
- **三维空间支持**：t_biz_target 表成功存储并解析了 POINT Z 三维坐标，证明 PostGIS 引擎已就绪，满足果园高度信息处理需求。
- **关系映射正确**：任务表与日志表准确关联了机器人与用户 ID，验证了底层外键约束及 ORM 一对多映射的完整性。
- **字段定义一致**：所有表项（含扩展字段如 image_url、password_hash）均被正确持久化，代码实现与物理模型设计文档完全一致。

6. 部署与运维设计

本章节基于“控制与决策子系统”作为协同系统核心中枢的定位 ， 构建物理体系结构模型。设计重点在于明确本子系统与外部 UAV/UGV 系统之间的**边界**、**接口**及**部署环境**。

6.1. 系统构件设计

根据 SRS 定义，本子系统负责接收异构数据并下发控制指令。因此，在构件设计中，我们将 UAV 和 UGV 视为**外部系统**，通过基础设施层的网关进行交互。

6.1.1. 核心构件说明

- **Coordination Control Core**
  - MissionManager：负责解析来自 UAV 的识别结果，生成任务列表。
  - AssignmentEngine：执行多目标优化的任务分配策略。
  - WorldState：维护全局环境与设备状态的数字孪生模型。
  - SafetyGuard：执行安全约束校验（如防碰撞、电量检测）。
- **Infrastructure Adapter (防腐层)**:
  - OpsGateway：作为系统的边界，封装 ROS 2 通信细节。它将内部指令转换为外部 UAV/UGV 能识别的标准消息（如 /cmd_vel 或 /goal_pose）。
- **External Interfaces (外部接口)**:
  - UAV Interface：订阅图像识别话题 (/uav/detection)。
  - UGV Interface：发布导航目标话题 (/ugv/goal) 并订阅状态话题 (/ugv/status)。

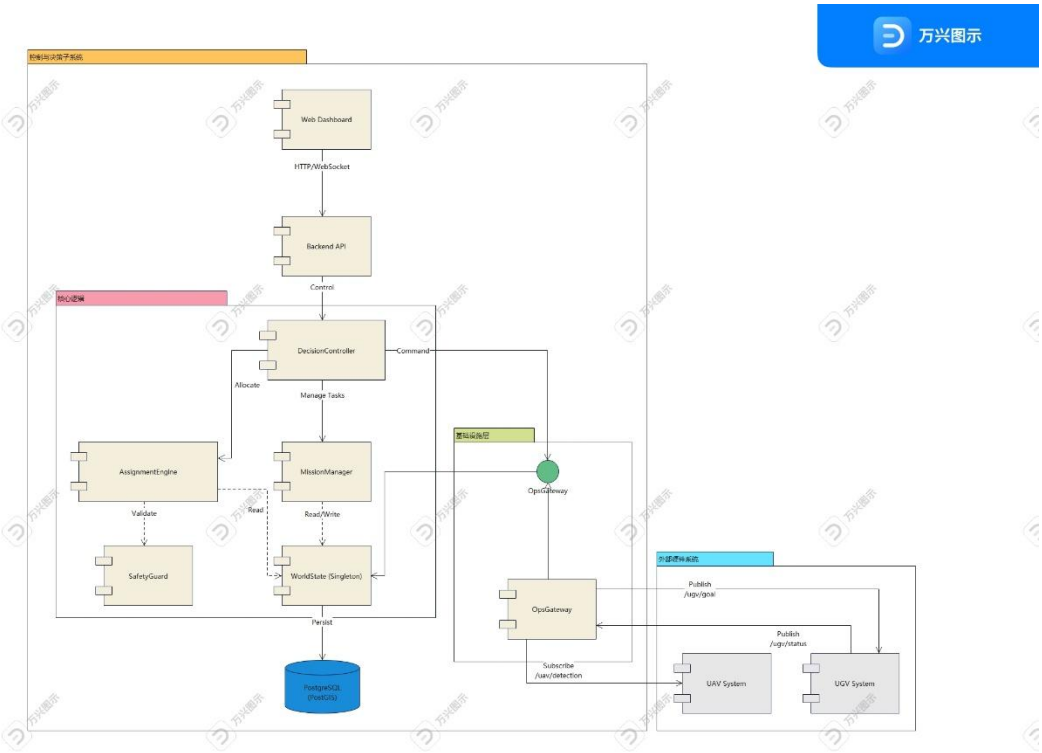

图 8 构件图

6.2. 物理部署设计

软件将部署在**地面控制站**或**高性能边缘服务器**上，通过无线网络远程指挥无人机和无人车。

6.2.1. 部署节点说明

- **地面控制服务器**:
  - **职责**：运行开发的“控制与决策子系统”。
  - **配置**：高性能笔记本或工控机（Ubuntu 22.04），部署 Docker 容器（包含 Web 后端、数据库、ROS 2 Master/Bridge）。
  - **理由**：SRS 要求处理 10Hz 图像数据和 50Hz 车辆状态，需要较强的算力支持，且不消耗 UAV/UGV 的机载电量。
- **外部节点**:
  - **UAV Node** 和 **UGV Node**：仅作为通过 Wi-Fi/5G 连接的外部设备出现在图中，表明物理连接关系，不展开其内部部署细节。
- **操作终端**:

- **职责**：管理员通过浏览器访问 Web 仪表盘。

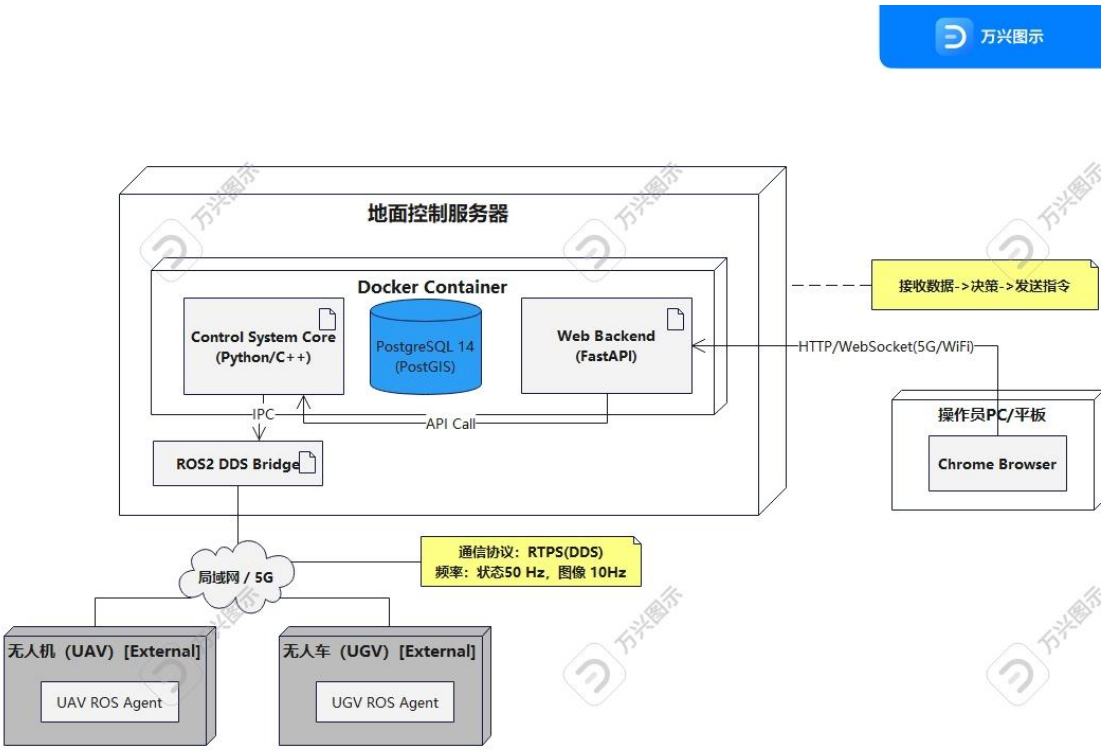

图 9 部署图

6.3. 运维设计

作为系统的控制中心，我的运维重点在于保障**数据的完整性**和**指令下发的可靠性**。

- **数据中心化日志**：
  - 所有外部 UAV/UGV 上报的异常（如 LOW_BATTERY），由我的 OpsGateway 统一接收并写入 t_sys_log 表。
  - **设计依据**：SRS 要求“记录分配结果”和“信息已记录”。
- **心跳监控机制**：
  - 我的 FaultMonitor 模块（在基础设施层）需每秒检查一次 t_sys_robot 表中的 last_heartbeat 字段。
  - **策略**：若 current_time - last_heartbeat > 5s，则判定该外部设备离线，自动触发重分配逻辑（参考 UC-004）。

7. 项目过程与配置管理

为了确保“无人机-无人车协同柑橘采摘系统”开发过程的规范性、可追溯性以及文档与代码的一致性，本项目采用了 **华为云 CodeArts (原 DevCloud)** 平台进行软件全生命周期管理（SDLC）。通过与 GitHub 的异构仓库协同，实现了从需求分析、架构建模到代码托管的闭环管理。

7.1. 需求规划与文档管理

本项目遵循敏捷开发理念，利用华为云的规划模块对《软件需求规格说明书》(SRS) 进行了数字化拆解与管理。

- **文档版本控制**：将 SRS 文档（软件需求规格说明书.docx）上传至文档库，确立了项目的基线需求，确保团队成员访问的始终是最新版本的需求文件。
- **需求条目化**：将 SRS 中的核心功能点（如“UC-001 分配采摘任务”、“UC-004 处理系统故障”）拆解为具体的 **Epic** 和 **Feature**，并通过看板视图进行可视化追踪，确保了需求实现的无遗漏。

图 10 华为云-1

7.2. 代码托管与双向同步

为了兼顾代码的安全存储与开源社区的协作便利性，本项目构建了“华为云 CodeHub + GitHub”的双仓库配置管理策略。

- **分布式版本控制**：源代码（包括后端 FastAPI、数据库脚本及前端页面）托管于华为云 CodeHub，严格遵循 Git Flow 工作流进行分支管理，确保主干代码的稳定性。
- **异构仓库协同**：通过配置远程仓库关联，实现了华为云 CodeHub 与 GitHub 的自动同步。此机制不仅提供了异地容灾备份，也验证了系统在不同代码托管环境下的适应性。

【华为云】:

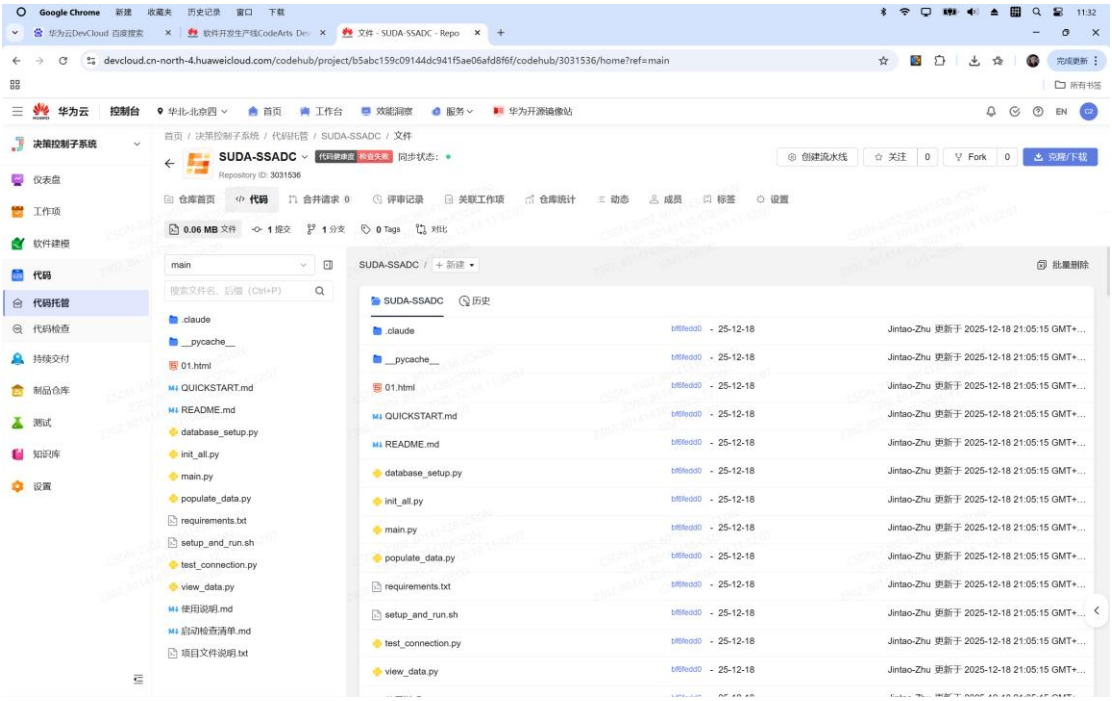

图 11 华为云-2

【GitHub】:


Jintao-Zhu readme
e2b2aa7 · 3 weeks ago
6 Commits
backend 12.1 3 weeks ago
frontend 12.1 3 weeks ago
README.md readme 3 weeks ago
README
极策云图 - 智慧果园图像检测系统
Citrus Vision Pro Intelligent System
一个基于 AI 视觉识别和协同调度的智慧果园管理系统
version 5.1.0 python 3.8+ vue 3.5+ license MIT
项目简介
极策云图（Citrus Vision Pro）是一个集成了 AI 图像识别、多智能体协同调度、任务排程和环境监控的智慧果园管理系统。系统采用前后端分离架构，支持无人机（UAV）和无人车（UGV）的协同作业，通过 YOLO 深度学习模型实现柑橘果实的智能检测与识别。
核心特性
AI 图像识别：基于 YOLO 模型的柑橘果实检测
协同调度：多智能体（UAV/UGV）路径规划与任务分配
任务排程：智能任务调度与执行管理
环境智控：果园环境参数监测与控制
知识库：农业百科与病虫害防治指南
策略引擎：自动化规则配置与执行
数据驾驶舱：可视化数据统计与分析
全域地图：基于 Leaflet 的地理信息展示

图 12 GitHub

图 13 技术栈说明

7.3. 在线建模与架构一致性

为了保证系统设计与实现的一致性，本项目利用华为云的建模功能，将报告中的 UML 模型进行了云端化管理。

- **模型可视化**：将本地设计的用例图、类图（如图 3-2）和时序图（如图 4-1）迁移至云端建模工具。这不仅便于在线评审，还支持通过模型直接生成部分代码框架，初步实现了模型驱动开发（MDD）的理念。
- **架构资产沉淀**：所有的 UML 模型与代码仓库深度绑定，确保了“设计文档”与“程序代码”的同步更新，解决了传统开发中“文档过时”的痛点。

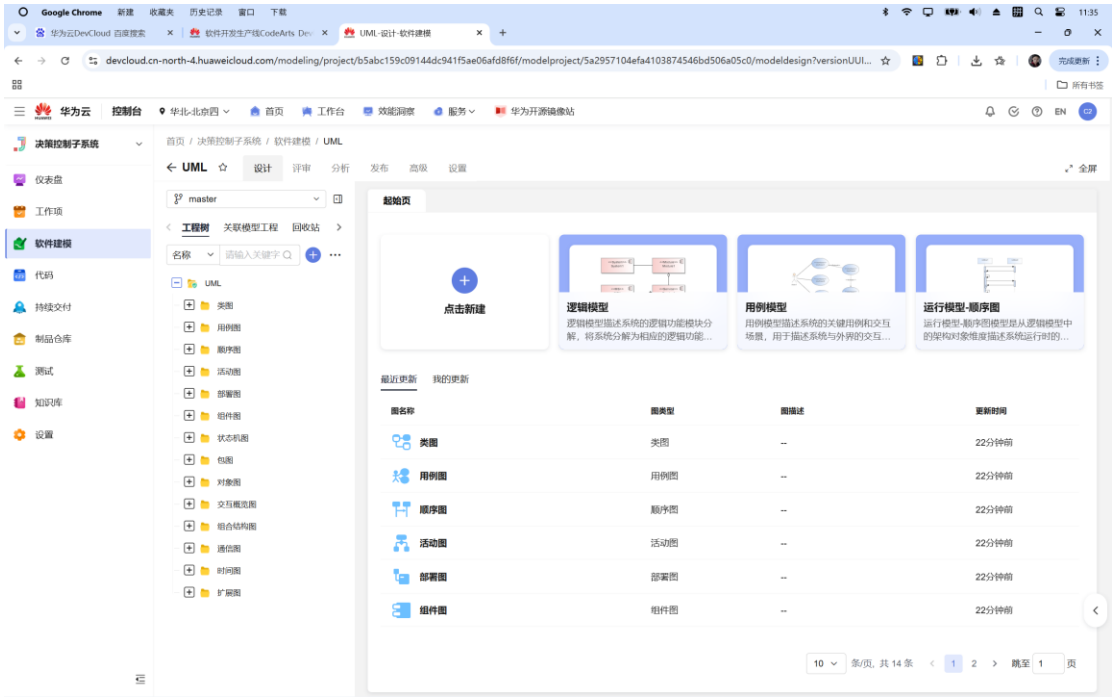

图 14 华为云-3

7.4. 过程管理总结

通过引入华为云 CodeArts 平台，本项目实现了一个规范的软件工程闭环：

- **需求可追溯**：从 SRS 文档到具体的代码提交均有迹可循。
- **资产更安全**：代码与文档实现了云端多重备份。
- **流程更规范**：遵循了行业标准的敏捷开发与配置管理流程。

这为系统的后续迭代、维护以及团队协作奠定了坚实的工程基础。

8. 总结与展望

8.1. 工作总结

本课程实践历时数周，通过对“控制与决策子系统”的完整设计与实现，我深刻理解了复杂软件系统的构建逻辑。回顾整个项目，主要成果总结如下：

- **架构设计落地**：成功构建了一个松耦合、高内聚的系统架构。通过引入 OpsGateway 防腐层，有效隔离了上层业务逻辑与底层硬件通信细节，使得系统能够灵活适配不

同的 UAV/UGV 设备。

- **核心算法实现**：在任务调度模块中，实现了基于优先级的任务分配策略，并预留了多目标优化接口，验证了“策略模式”在应对多变业务规则时的优越性。
- **数据底座构建**：利用 PostGIS 成功解决了三维空间坐标的存储与计算问题，为无人车的路径规划与避障提供了坚实的数据支撑，验证了数据库选型的正确性。
- **工程规范实践**：从 SRS 文档到 Git 提交记录，再到华为云上的敏捷看板，整个过程严格遵循软件工程规范，极大地提升了项目的可维护性与团队协作效率。

8.2. 存在不足

受限于时间与实验条件，本系统仍存在以下局限性：

- **算法复杂度有限**：目前的任务分配算法主要基于静态优先级规则，尚未引入遗传算法或强化学习等高级优化算法，在应对大规模车队（如 >20 台）时的调度效率有待验证。
- **仿真环境依赖**：目前的验证主要基于 WSL2 环境下的数据模拟，未接入真实的 Pixhawk 飞控或底盘硬件，实际通信中的丢包、延迟等网络抖动问题尚未充分考虑。

8.3. 未来展望

针对上述不足，未来计划从以下几个方面进行深入优化：

- **引入智能调度算法**：研究并集成多目标粒子群优化算法（MOPSO），在任务分配中同时考虑能耗、时间与设备损耗的平衡，实现全局最优解。
- **边缘计算与 5G 融合**：探索将 YOLOv11 推理模型部署在边缘端（如 Jetson Orin），利用 5G 网络的高带宽低时延特性，进一步提升系统的实时响应能力。
- **增强系统的容错性**：设计更完善的心跳监测与故障自愈机制（如基于 Paxos 的一致性算法），确保在单点故障发生时，协同系统仍能降级运行，保障作业安全。

通过不断的迭代与优化，期望该系统最终能从“实验室原型”走向“田间地头”，为智慧农业的发展贡献一份力量。

9. 参考文献与术语表

9.1. 参考文献

[1] Zhang, L., et al. "Multi-robot coordination for precision agriculture: A comprehensive survey." *Computers and Electronics in Agriculture*, vol. 185, 2021, pp. 106-119.
[2] 中华人民共和国农业农村部. 《农业机器人技术发展指导意见》. 农机发〔2022〕15 号, 2022.
[3] International Commission of Agricultural and Biosystems Engineering (CIGR). "Guidelines for autonomous agricultural robots." *CIGR Journal*, vol. 24, no. 2, 2022, pp. 15-28.
[4] Liu, S., et al. "Path planning and obstacle avoidance for autonomous agricultural robots: A review." *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, 2022, pp. 1825-1839.
[5] ISO 11783:2019. "Tractors and machinery for agriculture and forestry - Serial control and communications data network." International Organization for Standardization, 2019.
[6] Wang, H., et al. "Vision-based fruit recognition and harvesting robots in agriculture: A survey." *Biosystems Engineering*, vol. 215, 2022, pp. 261-281.
[7] 国家标准化管理委员会. GB/T 35487-2017 《农业机器人通用技术条件》. 中国标准出版社, 2017.
[8] Chen, Y., et al. "Real-time path planning for multi-robot systems in dynamic environments." *Robotics and Autonomous Systems*, vol. 142, 2021, pp. 103-115.

9.2. 术语表

术语/缩写	**中文全称**	**英文全称 (如有)**	**描述**
DWA	动态窗口算法	Dynamic Window Approach	一种用于局部避障规划的算法。
EKF	扩展卡尔曼滤波	Extended Kalman Filter	一种用于融合多传感器定位数据的算法。
GPS	全球定位系统	Global Positioning	用于获取无人车的位置坐

术语/缩写	中文全称	英文全称 (如有)	描述
		System	标信息。
IBVS	基于图像的视觉伺服	Image-Based Visual Servo	一种用于实现机械臂精确定位和抓取的控制方法。
IMU	惯性测量单元	Inertial Measurement Unit	用于获取无人车的姿态数据。
IP54	防护等级 54	Ingress Protection 54	衡量系统防尘防水能力的国际标准。
ISOBUS	国际农业机械标准 (ISO 11783)	(ISO 11783)	农业机械的串行控制和通信数据网络标准。
MTBF	平均故障间隔时间	Mean Time Between Failures	衡量系统可靠性的指标。
ROS	机器人操作系统	Robot Operating System	本系统设计遵循的机器人软件框架标准。
SD	时序图	Sequence Diagram	动态建模中用于交互分析的图表，如 SD-001。

术语/缩写	中文全称	英文全称 (如有)	描述
SLAM	(视觉)即时定位与地图构建	Simultaneous Localization and Mapping	用于在GPS信号弱时维持定位精度的技术。
STM	状态机	State Machine	动态建模中用于状态分析的图表，如 STM-Task。
UAV	无人机	Unmanned Aerial Vehicle	本协同系统中的空中单元，负责传输图像和识别结果。
UC	用例	Use Case	需求建模方法，用于描述系统的功能单元，如 UC-001。
UGV	无人车	Unmanned Ground Vehicle	本协同系统中的地面单元，负责执行路径规划和采摘。
UKF	无迹卡尔曼滤波	Unscented Kalman Filter	一种用于融合多传感器定位数据的算法。
YOLOv11	(YOLOv11) 目标	You Only Look	用于实时目标（柑橘）识

术语/缩写	中文全称	英文全称 (如有)	描述
	检测模型	Once v11	别的算法模型。

附录 A：核心代码实现清单

A.1 空间数据持久化模型 (ORM) 说明：展示了如何利用 SQLAlchemy 与 GeoAlchemy2 实现 PostGIS 空间数据类型的映射，以及数据库级约束的应用。

```
# 截取自 database_setup.py
class Target(Base):
    __tablename__ = 't_biz_target'

    # 使用 BigInteger 自增主键
    id = Column(BigInteger, primary_key=True, autoincrement=True)
    # 核心：映射 PostGIS 的 POINT Z 三维坐标类型
    coordinate = Column(Geometry(geometry_type='POINTZ', srid=4326),
nullable=False)
    # 数据库级约束：确保成熟度在 0.0 到 1.0 之间
    ripeness = Column(Float, CheckConstraint('ripeness >= 0 AND ripeness
<= 1.0'))
    image_url = Column(Text)
    area_code = Column(String(10))
```

A.2 任务调度与自动关联逻辑 说明：展示了后端 API 在创建任务时，如何自动关联目标坐标与用户信息，体现了业务逻辑的自动化。

```
# 截取自 main.py
@app.post("/api/tasks")
def create_task(task_data: TaskCreate, db: Session = Depends(get_db)):
    try:
        # 1. 自动根据区域代码生成三维空间坐标 (模拟 GIS 转换)
        coord_wkt = f'POINT Z({task_data.target_area_x}
{task_data.target_area_y} 1.5)'
        target = Target(
            coordinate=WKTElement(coord_wkt, srid=4326),
```

```
            ripeness=0.8,
            area_code=task_data.target_area
        )
        db.add(target)
        db.flush() # 获取生成的 ID

        # 2. 创建任务实体并建立外键关联
        new_task = Task(
            id=str(uuid.uuid4()),
            priority=task_data.priority,
            status="PENDING",
            target_id=target.id  # 关联上一步生成的空间目标
        )
        db.add(new_task)
        db.commit()
        return {"success": True, "task_id": new_task.id}
    except Exception as e:
        db.rollback()
        raise HTTPException(status_code=400, detail=str(e))
```

附录 B：API 接口文档概览

方法	资源路径	功能描述	核心参数
GET	/api/dashboard/stats	获取全局大屏统计数据	无
GET	/api/map/objects	获取 GIS 地图上的实体坐标	无 (返回 GeoJSON 格式)
POST	/api/tasks	创建新的协同采摘任务	target_area, priority
PUT	/api/tasks/{id}	更新任务状态或指派车辆	status, assigned_robot_id

方法	资源路径	功能描述	核心参数
POST	/api/robots	注册新的无人车设备	ip_address, battery_level

附录 C: 项目工程结构

说明: 本项目的代码组织遵循模块化原则, 实现了后端服务、数据库脚本与前端视图的分离。

```
ImageInspectionSystem/
├─── backend/                # 后端服务
│    ├─── app/
│    │    ├─── api/           # API 路由模块
│    │    │    ├─── auth.py     # 用户认证
│    │    │    ├─── dispatch.py # 协同调度
│    │    │    ├─── device.py   # 设备管理
│    │    │    ├─── schedule.py # 任务排程
│    │    │    ├─── environment.py # 环境智控
│    │    │    ├─── wiki.py     # 知识库
│    │    │    ├─── automation.py # 策略引擎
│    │    │    └─── ...
│    │    ├─── models/         # 数据模型
│    │    │    └─── schema.py   # 数据库表结构
│    │    ├─── services/       # 业务逻辑服务
│    │    │    └─── planning_service.py # 路径规划服务
│    │    ├─── utils/          # 工具函数
│    │    ├─── config.py       # 配置文件
│    │    └─── __init__.py     # 应用初始化
│    ├─── storage/            # 文件存储
│    │    ├─── originals/      # 原始图片
│    │    └─── processed/      # 处理后图片
│    ├─── init_db.py          # 数据库初始化脚本
│    ├─── run.py              # 应用启动入口
│    └─── citrus.db           # SQLite 数据库文件
│
├─── frontend/               # 前端应用
│    ├─── src/
│    │    ├─── api/           # API 接口封装
│    │    ├─── components/     # Vue 组件
│    │    ├─── views/          # 页面视图
```

```
│   │   │   ├──── DispatchView.vue    # 协同调度
│   │   │   ├──── MapView.vue         # 全域地图
│   │   │   ├──── ScheduleView.vue    # 任务排程
│   │   │   ├──── DashboardView.vue   # 数据驾驶舱
│   │   │   └──── ...
│   │   ├──── router/          # 路由配置
│   │   └──── main.js          # 应用入口
│   ├──── package.json
│   └──── vite.config.js
│
└──── README.md
```

GitHub 仓库：*https://github.com/Jintao-Zhu/The-first-software-copyright.git*